INTRODUCTION

In this project we have focused on building a machine learning (ML) model to predict the maliciousness of PDF files. We tried achieving the same by extracting relevant features from PDF files and utilizing multiple ML algorithms, the goal is to create a robust model capable of identifying potentially malicious content with a low false positive rate, and a high detection rate.

Why are we trying to achieve a low false positive rate?

We are trying to minimize false positive rate, as we do not want to misclassify a benign file as a malicious file, which might affect the reputation of the system. A false positive in a malicious file detector can trigger unnecessary alarms or actions, causing inconvenience, distrust, and potential disruption to normal operations. It can also lead to unnecessary investigation efforts and resource wastage, impacting user trust and system efficiency. Therefore, minimizing false positives is crucial to maintain user confidence, system credibility, and operational efficiency.

Using the few thousand labeled(benign , malicious) PDF data set provided, supervised learning is performed to train the ML model. After the model is trained, we use it to classify new PDF files provided as input, including those we have never seen in training. We measured the accuracy of the model using 75% of the dataset for training, and 25% for testing. When accuracy with four different ways of picking the 25% are averaged, also called cross validation.

FEATURE EXTRACTION

Now we have labeled files of both Malicious and Benign, so we have to extract features from these files as we cannot input the files as it is to the Machine Learning Model. For extracting the features we have used a script called **extract_feature** which accepts 3 parameters - input path, category and output path.

Command used are:

```
extract_feature --in /dataset/malicious --cat malicious --out features
extract_feature --in /dataset/benign --cat benign --out features
```

One for extracting features from malicious files and the other from benign files.

TYPES OF FEATURES:

On the broader terms we have 3 types of features that have been extracted:

1.YARA Signature: Yara is a powerful tool for pattern matching against binary files. Extracting Yara signatures involves identifying specific sequences or patterns within files that are indicative of known malware or suspicious behavior. Few YARA Signatures:

(i) "Without_attachments" suggests the absence of any files, documents, or content linked or enclosed within the PDF.

(ii) "With_urls" indicates the presence or inclusion of web addresses, links, or Uniform Resource Locators (URLs) within the PDF.

(iii) “invalid_trailer_structure" typically denotes an issue in the structure of a PDF file's trailer, indicating that the trailer dictionary, which contains essential metadata and pointers to other parts of the file, is improperly formatted or incomplete.

(iv) "Suspicious_creator" refers to a potentially dubious or unexpected value in the creator field of a PDF file's metadata, which might suggest irregularities or anomalies in the file's origin or creation details.

2.Static Properties: These are various attributes of files that can be determined without execution. Examples include file size, entropy, file type, presence of certain sections within a file, or metadata like creation/modification timestamps. Also Few more examples:

(i) XML forms in PDF files facilitate interactive data entry and user engagement, structured using XML elements for defining layout and content.

(ii) JBIG2Decode is an image compression algorithm primarily used for compressing bi-level (black and white) images in PDF documents.

(iii) "Launch_action" in a PDF refers to an action that triggers the launch of an application or script embedded within the PDF file, potentially indicating a security risk if not handled properly.

(iv) A “trailer dictionary” is a section located at the end of the file that contains essential information for reading and processing the PDF, such as pointers to important objects and cross-reference table details.

3.URIs (Uniform Resource Identifiers): URIs involve looking for URLs or web addresses embedded within files, which can sometimes indicate attempts to download additional malicious content or connect to malicious webpages.

PREPROCESSING:

We initialize by using the file paths for both benign and malicious PDFs located in google drive. We use this data for future preprocessing.

The preprocessing step involves various data extraction and transformation steps for feature creation and feature engineering and preparation.

- Extracting Yara Signatures: We extracted Yara signatures from both benign and malicious files, converting them into pandas dataframe structures.
- Extracting Static Properties: Static properties like metadata are gathered and organized into pandas dataframes for benign and malicious files.
- Extracting URXs: URXs, including URIs and URLs, are extracted from specific JSON keys in the provided files. Required domains are derived by comparing URXs between benign and malicious files.

In the next part of preprocessing we checked the existence of required domains in both benign and malicious files and stored the results in dictionaries and then converted the same into pandas dataframe.

In the final step, all the feature data frames are combined to create one final data frame one each for benign and malicious. Which is used in the next iteration to perform EDA and train the machine learning model.

EDA (Exploratory Data Analysis)

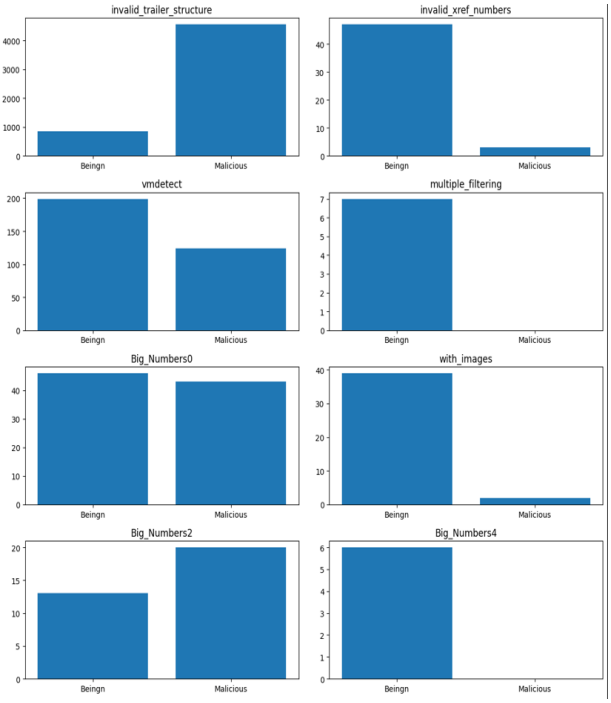
Data Collections: Obtaining the dataset from relevant data, i.e the preprocessed data frame that is generated in the previous step.

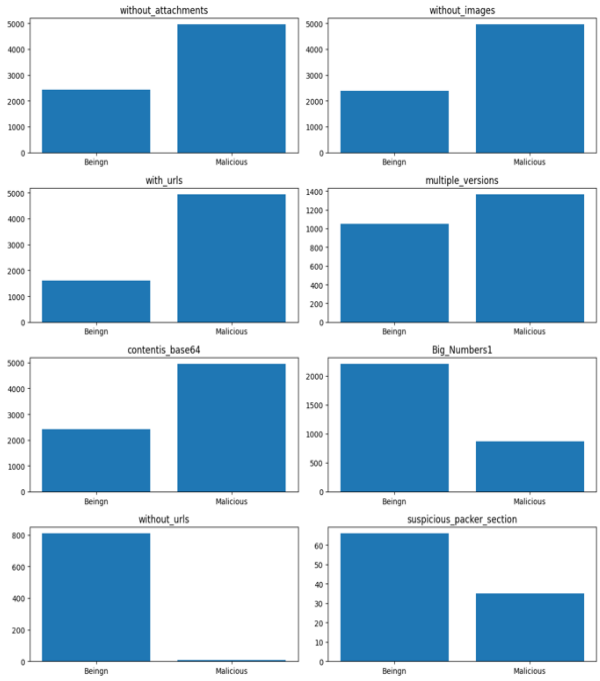
Data Cleaning: Addressed missing values by dropping them, converted data types of few features as it requires a particular data type, This step ensures data integrity.

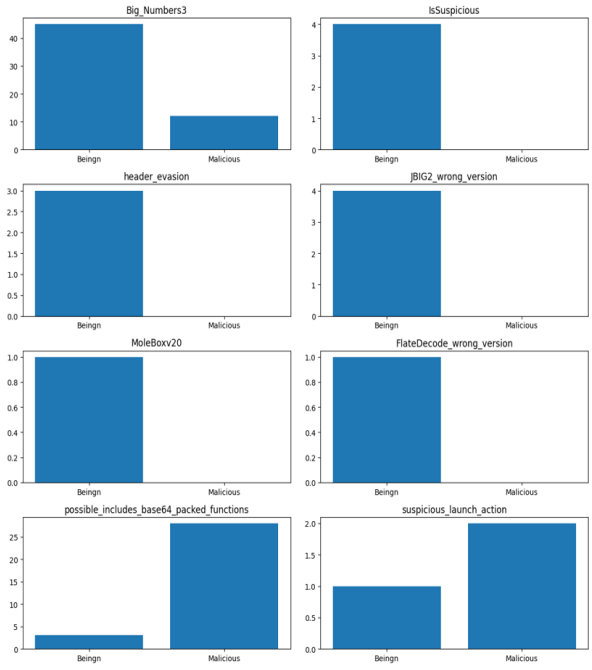
Feature Engineering: In this step one-hot encoding is performed to convert the categorical data types to convert them to numerical data type.

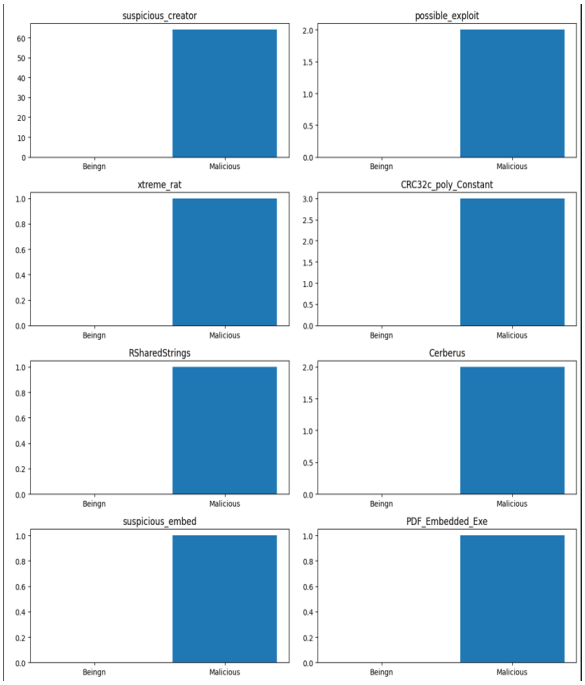
Visualization: Visualized the different distributions of the different features that were extracted from YARA properties and static properties, in each benign and malicious.

1. YARA Signature Distributions

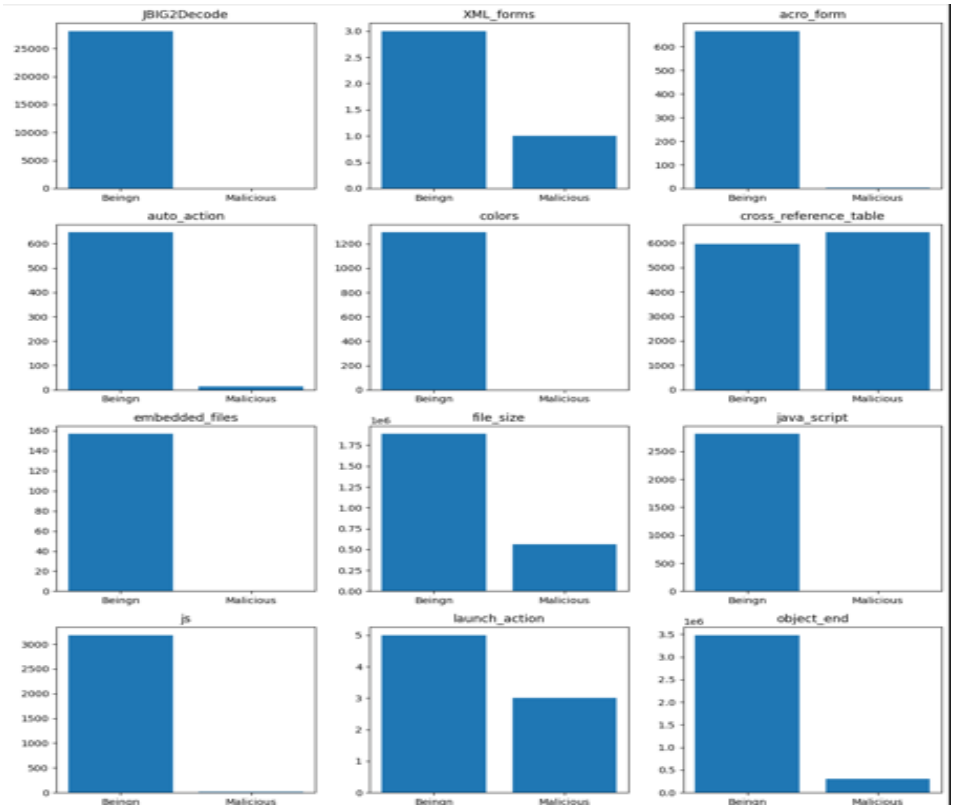


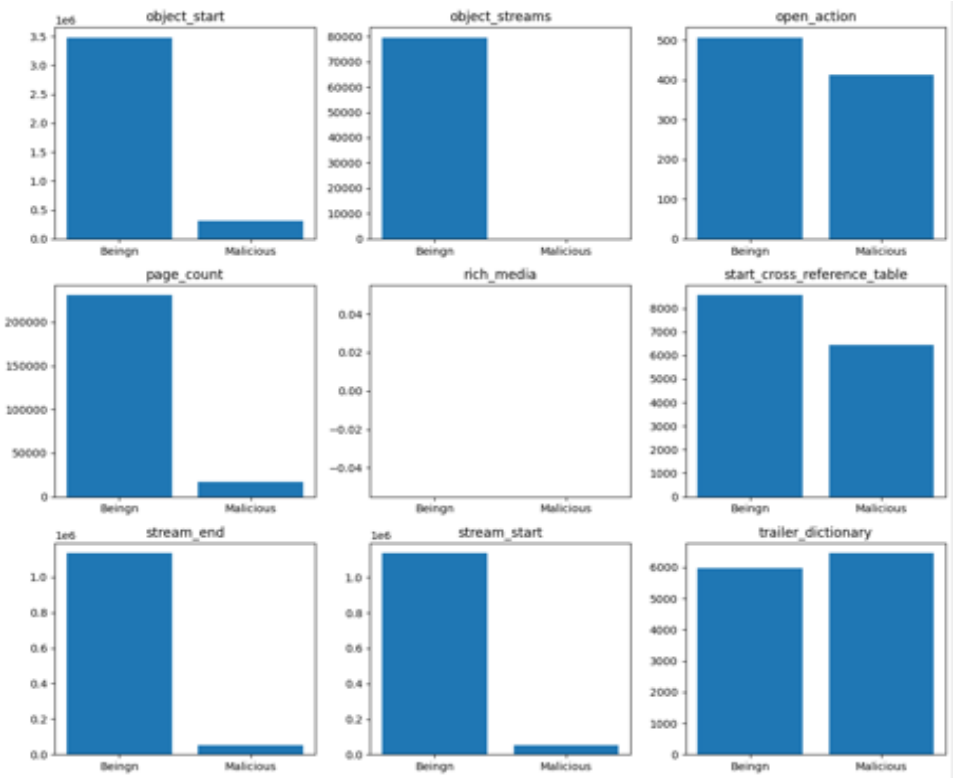




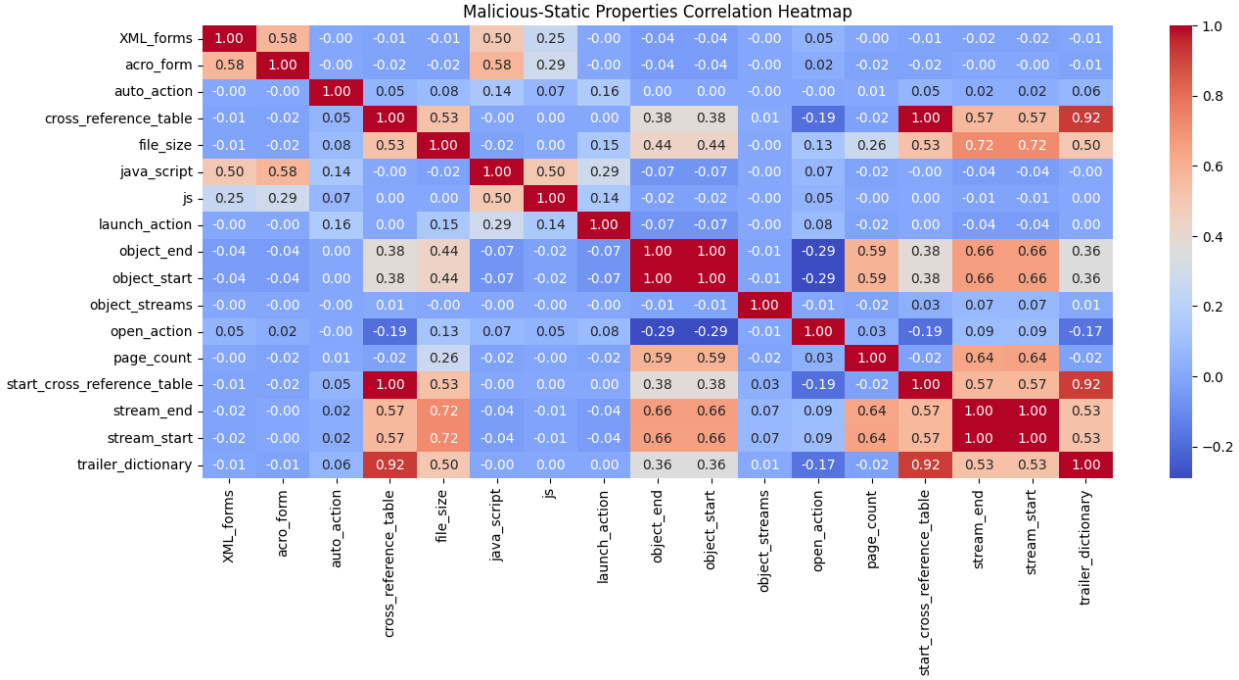


2. Static Properties Distribution

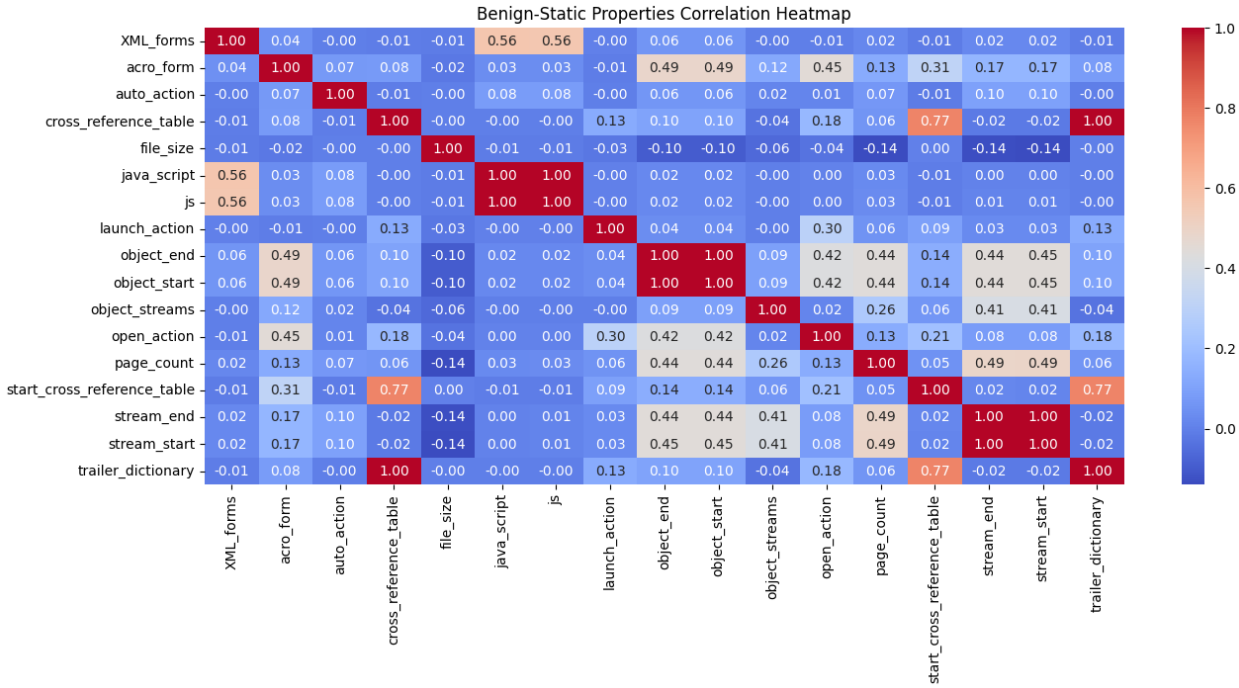




3. Malicious- Static Properties Correlation Heatmap



4. Benign- Static Properties Correlation Heatmap



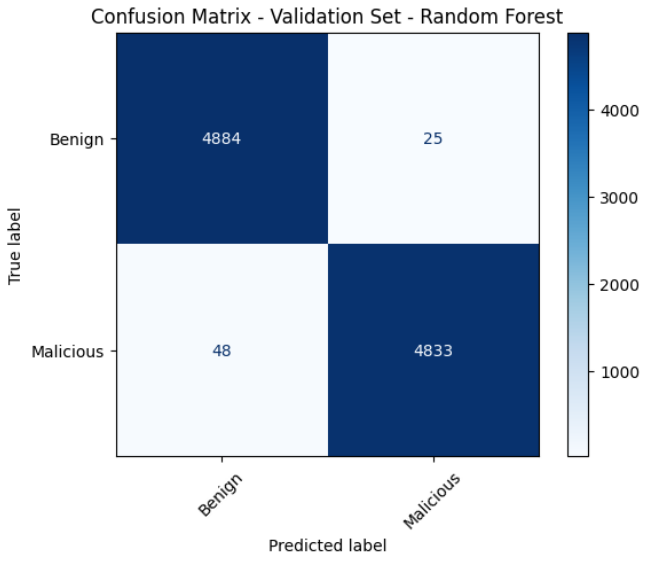
ML MODELS

1. Random Forest:

Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Each tree in the forest is constructed using a random subset of the dataset and a random subset of the features. It uses a technique called bagging (bootstrap aggregating) to build multiple decision trees and then combines their outputs to make predictions. Random Forest is known for its robustness, versatility, and ability to handle large datasets with high dimensionality.

By GridSearchCV we found the best params are **{'max_depth': 125, 'n_estimators': 111}**

Confusion Matrix: (4-Fold Cross Validation - 75% Train, 25% Test)



Accuracy - 99.24%, **False Positive** - 0.25%

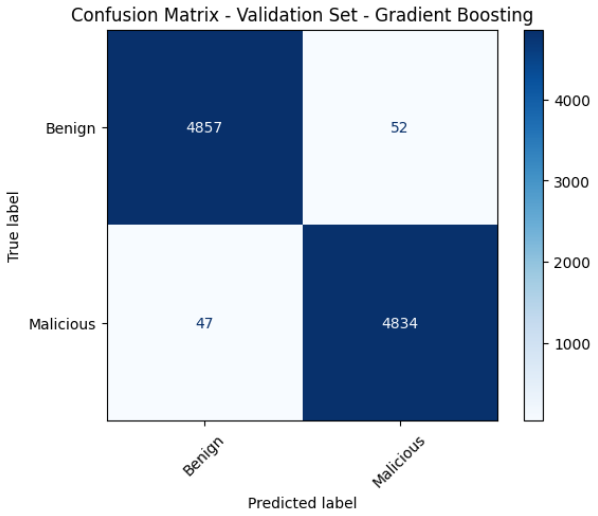
2. Gradient Boosting:

Gradient Boosting is another ensemble learning technique that builds a sequence of trees (typically decision trees) where each tree corrects the errors made by the previous one. Unlike Random Forest, which constructs multiple trees in parallel, Gradient Boosting builds trees sequentially. It focuses on minimizing a loss function by adding new models that predict the residuals or errors of the previous models. Gradient Boosting is effective in creating powerful

models by combining weak learners, and it's commonly used in regression and classification tasks.

By GridSearchCV we found the best params are **{'max_depth': 9, 'n_estimators': 111}**

Confusion Matrix: (4-Fold Cross Validation - 75% Train, 25% Test)



Accuracy - 98.98%, **False Positive** - 0.53%

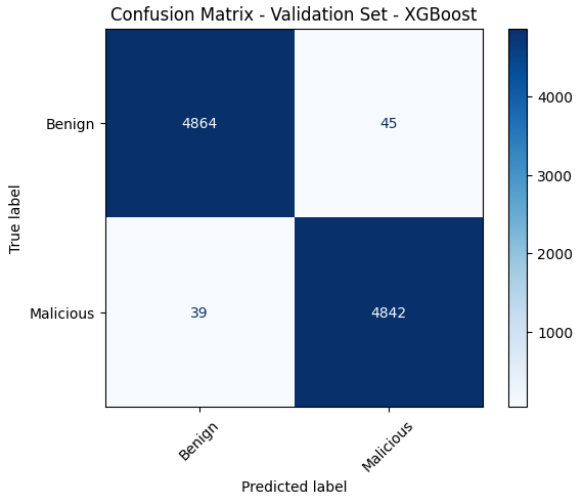
3. XGBoost (Extreme Gradient Boosting):

XGBoost is an optimized and scalable implementation of Gradient Boosting. It employs a more regularized model formalization to control overfitting and provides improved speed and performance over traditional Gradient Boosting. XGBoost uses a gradient descent algorithm for optimization and introduces a few additional features like parallel computing, tree pruning, and handling missing values, making it a popular choice in various machine learning competitions and real-world applications.

In all the ML Models the hyper parameters are asked as input from the user so that the user can test with multiple hyper parameters. We also performed grid search on each of them to get the best hyper parameters for each of the models.

By GridSearchCV we found the best params are **{'max_depth': 5, 'n_estimators': 350}**

Confusion Matrix: (4-Fold Cross Validation - 75% Train, 25% Test)

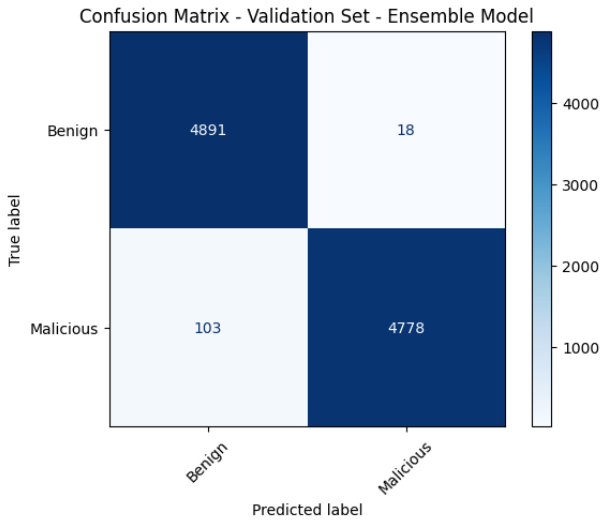


Accuracy - 99.14%, **False Positive** - 0.45%

4. Ensembling 5 Variants of Each of above 3 Models

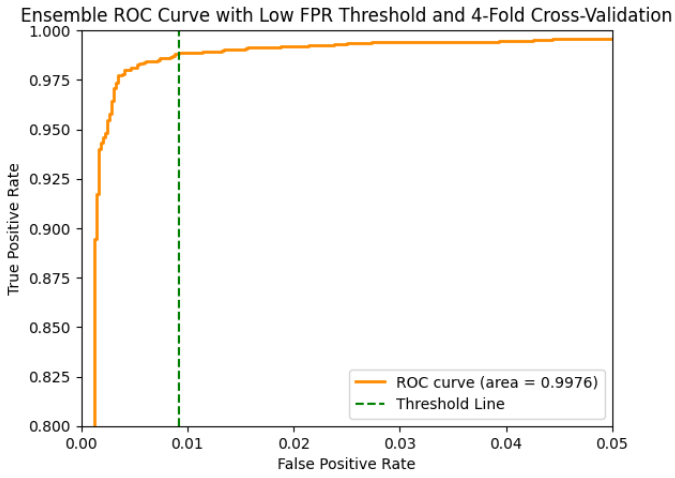
Now we tried ensembling 5 Variants of each of the 3 models Random Forest, Gradient Boosting and XGBoosting resulting in 15 Models, so to decrease the False Positives we took the condition for a file to be malicious in all 15 else we don;t consider it to be malicious and mark it benign. On top of this we did 4 Fold Cross Validation, to get accuracy.

Confusion Matrix: (4-Fold Cross Validation - 75% Train, 25% Test)



Accuracy - 98.76%, **False Positive** - 0.18%

ROC Curve (with threshold line)



DR v/s FP with Thresholds

```
DR vs FP with Thresholds
 0.000%     0.0000%          inf
 1.168%     0.0081%       0.9994
 4.466%     0.0081%       0.9994
 5.941%     0.0081%       0.9994
 7.171%     0.0081%       0.9993
 8.502%     0.0081%       0.9993
13.010%     0.0081%       0.9993
14.464%     0.0081%       0.9993
- -
- -
 89.428%     0.0122%       0.9870
 90.617%     0.0143%       0.9835
 92.030%     0.0163%       0.9781
 94.304%     0.0183%       0.9588
 95.103%     0.0244%       0.9438
 97.091%     0.0306%       0.8751
 97.726%     0.0346%       0.8130
 98.115%     0.0489%       0.7480
 98.422%     0.0631%       0.6350
 98.627%     0.0856%       0.5356
 98.873%     0.1039%       0.4404
 99.017%     0.1385%       0.3400
 99.140%     0.1813%       0.2377
```

So we found optimal threshold where we are having high Detection rate and Low False Positive by the formula at which threshold **TPR*(1-FPR)** is **maximum** and got the following results:

1. Detection Rate at Best Threshold - 98.853%
2. False Positive at Best Threshold - 0.0917%

Important Points:

1. Cross Validation makes better generalization, reduces overfitting bias, handles data variability and provides robust validation.
2. Working with potentially malicious content demands a higher ethical standard due to the potential risks involved. It's essential to prioritize privacy, fairness, transparency, and responsible use to mitigate harm and ensure that the benefits of any work with such content outweigh the risks. Collaboration with ethicists, legal experts, and impacted communities can help navigate these ethical considerations effectively.